

**What Is Claimed Is:**

1           1.       A method for using a hash table that is fully dynamic and lock-free,  
2 comprising:  
3           performing a lookup into the hash table, wherein the lookup involves,  
4                    using a hash key to lookup a bucket pointer in a bucket  
5                    array,  
6                    following the bucket pointer to a data node within a linked  
7                    list containing all of the data nodes in the hash table, and  
8                    searching from the data node through the linked list to  
9                    locate a node that matches the hash key if one exists;  
10          wherein the linked list contains only data nodes and at most a constant  
11 number of dummy nodes.

1           2.       The method of claim 1, wherein the data node pointed to by the  
2 bucket pointer precedes the nodes in the bucket.

1           3.       The method of claim 1, wherein deleting the data node from the  
2 linked list involves:  
3           using an atomic operation to mark the data node as dead; and  
4           atomically updating the next pointer of the predecessor of the data node to  
5 point around the data node to the successor of the data node in the linked list.

1           4.       The method of claim 2, wherein deleting the data node from the  
2 linked list additionally involves redirecting the next pointer of the data node to  
3 become a back pointer that points to the predecessor of the data node.

- 1           5.       The method of claim 4, wherein if a search through a chain of  
2 nodes from the back pointer does not lead to a live node, the method further  
3 comprises:  
4           obtaining a parent bucket pointer;  
5           searching through the linked list from a node pointed to by the parent  
6 bucket pointer to locate a starting node for the bucket pointer; and  
7           updating the bucket pointer to point to the starting node.
- 1           6.       The method of claim 2, wherein deleting the data node from the  
2 linked list involves using garbage collection or a solution to the repeat offender  
3 problem to reclaim the data node if possible.
- 1           7.       The method of claim 1, further comprising generating the hash key  
2 by performing a pre-hashing operation to achieve a uniform distribution of hash  
3 keys over possible hash key values.
- 1           8.       The method of claim 1, wherein if the average number of data  
2 nodes in each bucket exceeds a maximum value, the method further comprises:  
3           increasing the number of buckets in the bucket array to form a larger  
4 bucket array; and  
5           using more bits from the hash key to perform lookups in the larger bucket  
6 array.
- 1           9.       The method of claim 8, wherein buckets in the larger bucket array  
2 are initialized on-the-fly as they are referenced.

1           10.    The method of claim 8, wherein initializing a bucket pointer  
2 involves:  
3           obtaining a parent bucket pointer for the bucket pointer;  
4           searching through the linked list from a node pointed to by the parent  
5 bucket pointer to locate a starting node for the bucket pointer; and  
6           updating the bucket pointer to point to the starting node.

1           11.    The method of claim 1, wherein if there exists an old hash table,  
2 initializing a bucket pointer involves looking for a corresponding entry in the old  
3 hash table first, and if this fails:  
4           obtaining a parent bucket pointer for the bucket pointer;  
5           searching through the linked list from a node pointed to by the parent  
6 bucket pointer to locate a starting node for the bucket pointer; and  
7           updating the bucket pointer to point to the starting node.

1           12.    The method of claim 8,  
2 wherein the data nodes are stored in the linked list in bit-inverted hash key  
3 order; and  
4           wherein increasing the number of buckets in the bucket array involves  
5 mapping the existing bucket array into the top half of the larger bucket array.

1           13.    The method of claim 8,  
2 wherein the data nodes are stored in the linked list in hash key order; and  
3           wherein increasing the number of buckets in the bucket array involves  
4 interleaving the bucket array into the larger bucket array.

1           14.     The method of claim 2, wherein if the average number of data  
2 nodes in each bucket falls below a minimum value, the method further comprises:  
3           reducing the number of buckets in the bucket array to form a smaller  
4 bucket array; and  
5           using fewer bits from the hash key to perform lookups in the smaller  
6 bucket array.

1           15.     A computer-readable storage medium storing instructions that  
2 when executed by a computer cause the computer to perform a method for using a  
3 hash table that is fully dynamic and lock-free, the method comprising:  
4           performing a lookup into the hash table, wherein the lookup involves,  
5                    using a hash key to lookup a bucket pointer in a bucket  
6                    array,  
7                    following the bucket pointer to a data node within a linked  
8                    list containing all of the data nodes in the hash table, and  
9                    searching from the data node through the linked list to  
10                  locate a node that matches the hash key if one exists;  
11                  wherein the linked list contains only data nodes and at most a  
12 constant number of dummy nodes.

1           16.     The computer-readable storage medium of claim 15, wherein the  
2 data node pointed to by the bucket pointer precedes the nodes in the bucket.

1           17.     The computer-readable storage medium of claim 15, wherein  
2 deleting the data node from the linked list involves:  
3           using an atomic operation to mark the data node as dead; and

4           atomically updating the next pointer of the predecessor of the data node to  
5   point around the data node to the successor of the data node in the linked list.

1           18.    The computer-readable storage medium of claim 16, wherein  
2   deleting the data node from the linked list additionally involves redirecting the  
3   next pointer of the data node to become a back pointer that points to the  
4   predecessor of the data node.

1           19.    The computer-readable storage medium of claim 18, wherein if a  
2   search through a chain of nodes from the back pointer does not lead to a live node,  
3   the method further comprises:  
4       obtaining a parent bucket pointer;  
5       searching through the linked list from a node pointed to by the parent  
6   bucket pointer to locate a starting node for the bucket pointer; and  
7       updating the bucket pointer to point to the starting node.

1           20.    The computer-readable storage medium of claim 16, wherein  
2   deleting the data node from the linked list involves using garbage collection or a  
3   solution to the repeat offender problem to reclaim the data node if possible.

1           21.    The computer-readable storage medium of claim 15, wherein the  
2   method further comprises generating the hash key by performing a pre-hashing  
3   operation to achieve a uniform distribution of hash keys over possible hash key  
4   values.

1           22.    The computer-readable storage medium of claim 15, wherein if the  
2   average number of data nodes in each bucket exceeds a maximum value, the  
3   method further comprises:  
4           increasing the number of buckets in the bucket array to form a larger  
5   bucket array; and  
6           using additional bits from the hash key to perform lookups in the larger  
7   bucket array.

1           23.    The computer-readable storage medium of claim 22, wherein  
2   buckets in the larger bucket array are initialized on-the-fly as they are referenced.

1           24.    The computer-readable storage medium of claim 22, wherein  
2   initializing a bucket pointer involves:  
3           obtaining a parent bucket pointer for the bucket pointer;  
4           searching through the linked list from a node pointed to by the parent  
5   bucket pointer to locate a starting node for the bucket pointer; and  
6           updating the bucket pointer to point to the starting node.

1           25.    The computer-readable storage medium of claim 15, wherein if  
2   there exists an old hash table, initializing a bucket pointer involves looking for a  
3   corresponding entry in the old hash table first, and if this fails:  
4           obtaining a parent bucket pointer for the bucket pointer;  
5           searching through the linked list from a node pointed to by the parent  
6   bucket pointer to locate a starting node for the bucket pointer; and  
7           updating the bucket pointer to point to the starting node.

1           26.    The computer-readable storage medium of claim 22,

2            wherein the data nodes are stored in the linked list in bit-inverted hash key  
3 order; and

4            wherein increasing the number of buckets in the bucket array involves  
5 mapping the existing bucket array into the top half of the larger bucket array.

1            27.    The computer-readable storage medium of claim 22,  
2            wherein the data nodes are stored in the linked list in hash key order; and  
3            wherein increasing the number of buckets in the bucket array involves  
4 interleaving the bucket array into the larger bucket array.

1            28.    The computer-readable storage medium of claim 15, wherein if the  
2 average number of data nodes in each bucket falls below a minimum value, the  
3 method further comprises:

4            reducing the number of buckets in the bucket array to form a smaller  
5 bucket array; and

6            using less bits from the hash key to perform lookups in the smaller bucket  
7 array.

1            29.    An apparatus that implements a hash table that is fully dynamic  
2 and lock-free, comprising:

3            a lookup mechanism, wherein the lookup mechanism is configured to,  
4                            use a hash key to lookup a bucket pointer in a bucket array,  
5                            follow the bucket pointer to a data node within a linked list  
6                            containing all of the data nodes in the hash table, and to  
7                            search from the data node through the linked list to locate a  
8                            node that matches the hash key if one exists;

9                    wherein the linked list contains only data nodes and at most a  
10       constant number of dummy nodes.

1            30.     The apparatus of claim 29, wherein the data node pointed to by the  
2       bucket pointer precedes the nodes in the bucket.

1            31.     The apparatus of claim 29, further comprising a node deletion  
2       mechanism configured to:  
3            use an atomic operation to mark the data node as dead; and to  
4            atomically update the next pointer of the predecessor of the data node to  
5       point around the data node to the successor of the data node in the linked list.

1            32.     The apparatus of claim 30, wherein the node deletion mechanism is  
2       additionally configured to redirect the next pointer of the data node to become a  
3       back pointer that points to the predecessor of the data node.

1            33.     The apparatus of claim 32, further comprising a bucket pointer  
2       updating mechanism, wherein if a search through a chain of nodes from the back  
3       pointer does not lead to a live node, the bucket pointer updating mechanism is  
4       configured to:  
5            obtain a parent bucket pointer;  
6            search through the linked list from a node pointed to by the parent bucket  
7       pointer to locate a starting node for the bucket pointer; and to  
8            update the bucket pointer to point to the starting node.



1           34.     The apparatus of claim 30, wherein the node deletion mechanism is  
2 additionally configured to use garbage collection or a solution to the repeat  
3 offender problem to reclaim the data node if possible.

1           35.     The apparatus of claim 29, further comprising a pre-hashing  
2 mechanism configured to generate the hash key by performing a pre-hashing  
3 operation to achieve a uniform distribution of hash keys over possible hash key  
4 values.

1           36.     The apparatus of claim 29, further comprising a bucket array  
2 expansion mechanism, wherein if the average number of data nodes in each  
3 bucket exceeds a maximum value, the bucket array expansion mechanism is  
4 configured to:  
5           increase the number of buckets in the bucket array to form a larger bucket  
6 array; and to  
7           use additional bits from the hash key to perform lookups in the larger  
8 bucket array.

1           37.     The apparatus of claim 36, further comprising a bucket  
2 initialization mechanism configured to initialize buckets in the larger bucket array  
3 on-the-fly as they are referenced.

1           38.     The apparatus of claim 36, wherein the bucket initialization  
2 mechanism is configured to:  
3           obtain a parent bucket pointer for the bucket pointer;  
4           search through the linked list from a node pointed to by the parent bucket  
5 pointer to locate a starting node for the bucket pointer; and to

6           update the bucket pointer to point to the starting node.

1           39.     The apparatus of claim 29, wherein if there exists an old hash  
2     table, the bucket initialization mechanism is configured to look for a  
3     corresponding entry in the old hash table first, and if this fails to:  
4           obtain a parent bucket pointer for the bucket pointer;  
5           search through the linked list from a node pointed to by the parent bucket  
6     pointer to locate a starting node for the bucket pointer; and to  
7           update the bucket pointer to point to the starting node.

1           40.     The apparatus of claim 36,  
2           wherein the data nodes are stored in the linked list in bit-inverted hash key  
3     order; and  
4           wherein the bucket array expansion mechanism is configured to map the  
5     existing bucket array into the top half of the larger bucket array.

1           41.     The apparatus of claim 36,  
2           wherein the data nodes are stored in the linked list in hash key order; and  
3           wherein the bucket array expansion mechanism is configured to interleave  
4     the bucket array into the larger bucket array.

1           42.     The apparatus of claim 30, further comprising a bucket array  
2     contraction mechanism, wherein if the average number of data nodes in each  
3     bucket falls below a minimum value, the bucket array contraction mechanism is  
4     configured to:  
5           reduce the number of buckets in the bucket array to form a smaller bucket  
6     array; and to

7            use less bits from the hash key to perform lookups in the smaller bucket  
8    array.